# LAB 4

## PART A: AGGREGATE FUNCTIONS

## AVG( )

SQL uses the AVG() function to calculate the average of a column. The syntax for using this function is,

```
SELECT AVG("column_name")
FROM "table_name"
```

For example, if we want to get the average of all sales from the following table,

Table ***Store_Information***

| store_name | Sales | Date |
|---|---|---|
| Los Angeles | $1500 | Jan-05-1999 |
| San Diego | $250 | Jan-07-1999 |
| Los Angeles | $300 | Jan-08-1999 |
| Boston | $700 | Jan-08-1999 |

we would type in

```
SELECT AVG(Sales) FROM Store_Information
```

*Result:*

| AVG(Sales) |
|---|
| $687.5 |

$687.5 represents the average of all Sales entries: ($1500 + $250 + $300 + $700) / 4.

## COUNT( )

Another arithmetic function is **COUNT**. This allows us to **COUNT** up the number of row in a certain table. The syntax is,

```
SELECT COUNT("column_name")
FROM "table_name"
```

For example, if we want to find the number of store entries in our table,

Table ***Store_Information***

| store_name | Sales | Date |
|---|---|---|
| Los Angeles | $1500 | Jan-05-1999 |
| San Diego | $250 | Jan-07-1999 |
| Los Angeles | $300 | Jan-08-1999 |
| Boston | $700 | Jan-08-1999 |

we'd key in

```
SELECT COUNT(store_name)
FROM Store_Information
```

*Result:*

| Count(store_name) |
|---|
| 4 |

**COUNT** and **DISTINCT** can be used together in a statement to fetch the number of distinct entries in a table. For example, if we want to find out the number of distinct stores, we'd type,

```
SELECT COUNT(DISTINCT store_name)
FROM Store_Information
```

*Result:*

| Count(DISTINCT store_name) |
|---|
| 3 |

## MAX ( )

SQL uses the MAX function to find the maximum value in a column. The syntax for using the MAX function is,

```
SELECT MAX("column_name")
FROM "table_name"
```

For example, if we want to get the highest sales from the following table,

Table ***Store_Information***

| store_name | Sales | Date |
|---|---|---|
| Los Angeles | $1500 | Jan-05-1999 |
| San Diego | $250 | Jan-07-1999 |
| Los Angeles | $300 | Jan-08-1999 |
| Boston | $700 | Jan-08-1999 |

we would type in

**SELECT MAX(Sales) FROM Store_Information**

*Result:*

**MAX(Sales)**
**$1500**

$1500 represents the maximum value of all Sales entries: $1500, $250, $300, and $700.

## MIN ( )

SQL uses the MIN function to find the maximum value in a column. The syntax for using the MIN function is,

```
SELECT MIN("column_name")
FROM "table_name"
```

For example, if we want to get the lowest sales from the following table,

Table ***Store_Information***

| store_name | Sales | Date |
|---|---|---|
| Los Angeles | $1500 | Jan-05-1999 |
| San Diego | $250 | Jan-07-1999 |
| Los Angeles | $300 | Jan-08-1999 |
| Boston | $700 | Jan-08-1999 |

we would type in

**SELECT MIN(Sales) FROM Store_Information**

*Result:*

**MIN(Sales)**
**$250**

$250 represents the minimum value of all Sales entries: $1500, $250, $300, and $700.

SQL uses the MIN function to find the maximum value in a column. The syntax for using the MIN function is,

```
SELECT MIN("column_name")
FROM "table_name"
```

For example, if we want to get the lowest sales from the following table,

Table ***Store_Information***

| store_name | Sales | Date |
|---|---|---|
| Los Angeles | $1500 | Jan-05-1999 |
| San Diego | $250 | Jan-07-1999 |
| Los Angeles | $300 | Jan-08-1999 |
| Boston | $700 | Jan-08-1999 |

we would type in

```
SELECT MIN(Sales) FROM Store_Information
```

*Result:*

| MIN(Sales) |
|---|
| $250 |

$250 represents the minimum value of all Sales entries: $1500, $250, $300, and $700.

## SUM( )

The SUM function is used to calculate the total for a column. The syntax is,

```
SELECT SUM("column_name")
FROM "table_name"
```

For example, if we want to get the sum of all sales from the following table,

Table ***Store_Information***

| store_name | Sales | Date |
|---|---|---|
| Los Angeles | $1500 | Jan-05-1999 |
| San Diego | $250 | Jan-07-1999 |
| Los Angeles | $300 | Jan-08-1999 |

| | | |
|---|---|---|
| Boston | $700 | Jan-08-1999 |

we would type in

**SELECT SUM(Sales) FROM Store_Information**

*Result:*

| SUM(Sales) |
|---|
| $2750 |

$2750 represents the sum of all Sales entries: $1500 + $250 + $300 + $700

## Exercise Part A:

1. Calculate the average credit limit of the customers

SELECT AVG(CR_LIMIT) FROM CUSTOMERS;\

| AVG(CR_LIMIT) |
|---|
| 940 |

2. Who has the lowest current balance among the list of the customers?

   SELECT MIN(CURR_BALANCE) FROM CUSTOMERS;

| MIN(CURR_BALANCE) |
|---|
| 0 |

3. What is the average credit limit of customers staying in Mill Park?

   SELECT AVG(CR_LIMIT)
   FROM CUSTOMERS
   WHERE TOWN = 'Mill Park';

| AVG(CR_LIMIT) |
|---|
| 650 |

4. Display the highest, lowest, sum and average credit limit of all customers. Label the columns Maximum, Minimum, Sum and Average, respectively.

   SELECT MAX(CR_LIMIT) MAXIMUM , MIN(CR_LIMIT) MINIMUM ,
   SUM(CR_LIMIT) SUM, AVG(CR_LIMIT) AVERAGE
   FROM CUSTOMERS;

| MAXIMUM | MINIMUM | SUM | AVERAGE |
|---|---|---|---|
| 2800 | 300 | 4700 | 940 |

5. Write a query to display the difference between the highest and lowest current balance. Label the column 'Difference'.

```
SELECT (MAX(CR_LIMIT) - MIN(CR_LIMIT)) Difference
from Customers;
```

| DIFFERENCE |
|---|
| 2500 |

6. How many customers are staying in Mill Park?

```
SELECT COUNT(TOWN)
FROM CUSTOMERS
WHERE TOWN = 'MILL PARK';
```

| COUNT(TOWN) |
|---|
| 2 |

7. Calculate the total number of orders being made by the customers.

```
SELECT COUNT(ORDER_NO)
FROM ORDERS;
```

| COUNT(ORDER_NO) |
|---|
| 3 |

8. How many customers are having a credit limit of RM300 and staying in Mill Park

```
SELECT COUNT(CUST_NO)
FROM CUSTOMERS
WHERE CR_LIMIT = 300
AND
TOWN = 'MILL PARK';
```

| COUNT(CUST_NO) |
|---|
| 0 |

# CREATE TABLE

Tables are the basic structure where data is stored in the database. Given that in most cases, there is no way for the database vendor to know ahead of time what your data storage needs are, chances are that you will need to create tables in the database yourself. Many database tools allow you to create tables without writing SQL, but given that tables are the container of all the data, it is important to include the **CREATE TABLE** syntax in this tutorial.

Before we dive into the SQL syntax for **CREATE TABLE**, it is a good idea to understand what goes into a table. Tables are divided into rows and columns. Each row represents one piece of data, and each column can be thought of as representing a component of that piece of data. So, for example, if we have a table for recording customer information, then the columns may include information such as First Name, Last Name, Address, City, Country, Birth Date, and so on. As a result, when we specify a table, we include the column headers and the data types for that particular column.

So what are data types? Typically, data comes in a variety of forms. It could be an integer (such as 1), a real number (such as 0.55), a string (such as 'sql'), a date/time expression (such as '2000-JAN-25 03:22:22'), or even in binary format. When we specify a table, we need to specify the data type associated with each column (i.e., we will specify that 'First Name' is of type char(50) - meaning it is a string with 50 characters). One thing to note is that different relational databases allow for different data types, so it is wise to consult with a database-specific reference first.

The SQL syntax for **CREATE TABLE** is

```
CREATE TABLE "TABLE_NAME"
("COLUMN 1" "DATA_TYPE_FOR_COLUMN_1",
"COLUMN 2" "DATA_TYPE_FOR_COLUMN_2",
... )
```

So, if we are to create the customer table specified as above, we would type in

```
CREATE TABLE customer
(First_Name char(50),
Last_Name char(50),
Address char(50),
City char(50),
Country char(25),
Birth_Date date)
```

Sometimes, we want to provide a default value for each column. A default value is used when you do not specify a column's value when inserting data into the table. To specify a default value, add "Default [value]" after the data type declaration. In the above example, if we want to default column "Address" to "Unknown" and City to "Mumbai", we would type in

```
CREATE TABLE customer
(First_Name char(50),
Last_Name char(50),
Address char(50) default 'Unknown',
City char(50) default 'Mumbai',
Country char(25),
Birth_Date date)
```

# CONSTRAINT

You can place constraints to limit the type of data that can go into a table. Such constraints can be specified when the table when the table is first created via the **CREATE TABLE** statement, or after the table is already created via the **ALTER TABLE** statement.

Common types of constraints include the following:

- **NOT NULL Constraint**: Ensures that a column cannot have NULL value.
- **DEFAULT Constraint**: Provides a default value for a column when none is specified.
- **UNIQUE Constraint**: Ensures that all values in a column are different.
- **CHECK Constraint**: Makes sure that all values in a column satisfy certain criteria.
- **Primary Key Constraint**: Used to uniquely identify a row in the table.
- **Foreign Key Constraint**: Used to ensure referential integrity of the data.

Each constraint is discussed in the following sections.

```
CREATE TABLE STUDENT_COURSE
(
    STUDENT_ID NUMBER,
    STUDENT_NAME VARCHAR(2),
    COURSE_ID NUMBER,
    SEMESTER NUMBER
);
```

DESCRIBE STUDENT_COURSE;

| Name | Null? | Type |
|---|---|---|
| STUDENT_ID | | NUMBER |
| COURSE_ID | | NUMBER |
| SEMESTER | | NUMBER |
| STUDENT_NAME | | VARCHAR2(10) |